Neeraj Appai

Date 13/07/21

SDS Page No.
Date / /

A.7

Practical - 02

Aim - Write a program in python to implement iterative deep depth first search for romanian map problem

* IDFS Description -
Iterative Deepening depth first search is a general strategy, often used in combination with depth first search, that finds the best depth limit
* Strategies are evaluated along the following dimensions
a) Completeness : Does it always find a solution if one exists ?
-> This algorithm is complete it is if the branching factor b is finite
b) Optamality - Does it always find a ~~solution if or~~ ex a least - count solution ?
-> The algorithm is optimal if path cost is a non decreasing function of depth of the node
c) time complexity : Number of nodes generated ?
-> The total number of nodes generated are N(IDS) $= (d)b + (d-1)b^2 + \ldots + 1(b)^d$, which gives a time complexity of $O(b^d)$
d) Space complexity : Maximum number of nods in memory
-> The space complexity of IDFS will be $O(bd)$ and it does not have enough memory
e) Time and space complexity are measured in terms of the time is often measured in terms of number of nodes during the search and space in terms of maximum number of nodes stored in memory

Nemoy Appa ~~the~~

1) b. The branching factor of maximum number of successor of any node

2) d. depth of the least cost solution
The depth of the shallowest goal node ie the number of along the path of root

3) m. maximum depth of the state space (maybe)
The maximum lenght of any path in state space

4) Search cost (time) total cost (time, space) the path cost of the cost solution found which typically based in on the complexity but on also include a term for memory usage or can use total cost which combines search cost and path cost of the solution found

2) IDFS Algorithm

1) from collections import deque
2) use infinite class graph and add functions with self _init_ and get
3) In ~~a~~ Add class problem with object parameter which is used to define functions like _init_, actions, result, goal_test, path_cost and value
4) Take infinity = float('inf')
5) Take class Node
6) add functions with init which has self.state; self.parent etc and add self.depth = parent.depth + 1
7) ~~A~~ Define functions like __repr__, expand, child node which is used to find next state,

Neeraj Appar

new cost and next_node

8) def path and solution

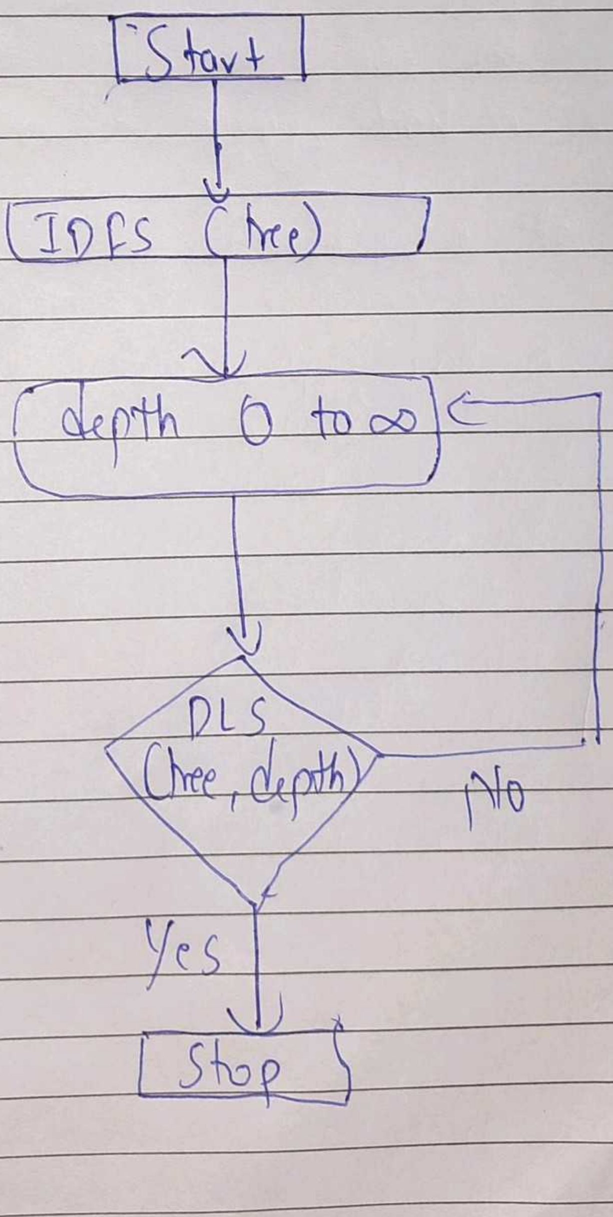
9) def iterative_deepening_search and write the algorithm

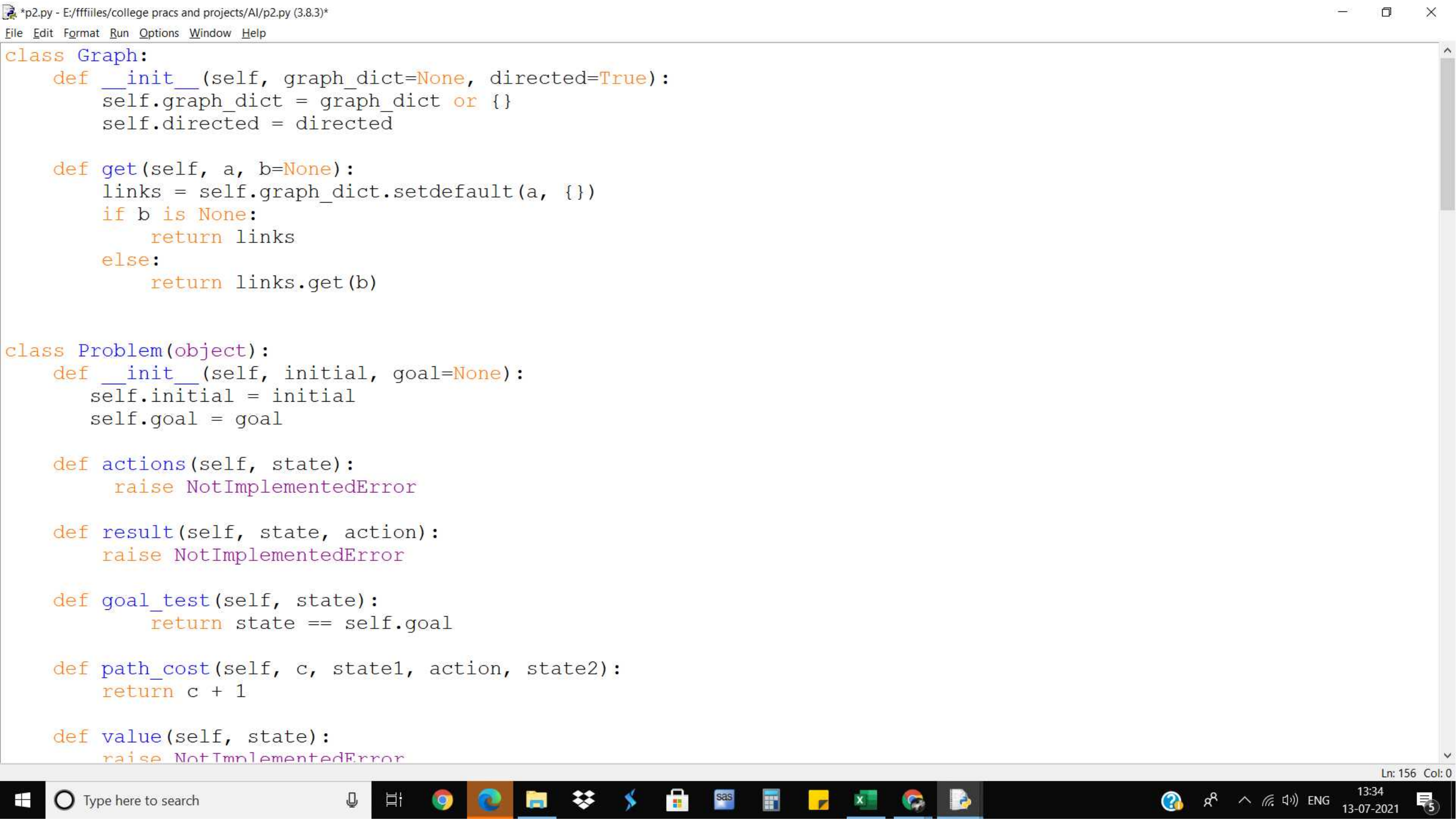
10) ~~Print~~ give the inputs to the map

11) Print the romania map

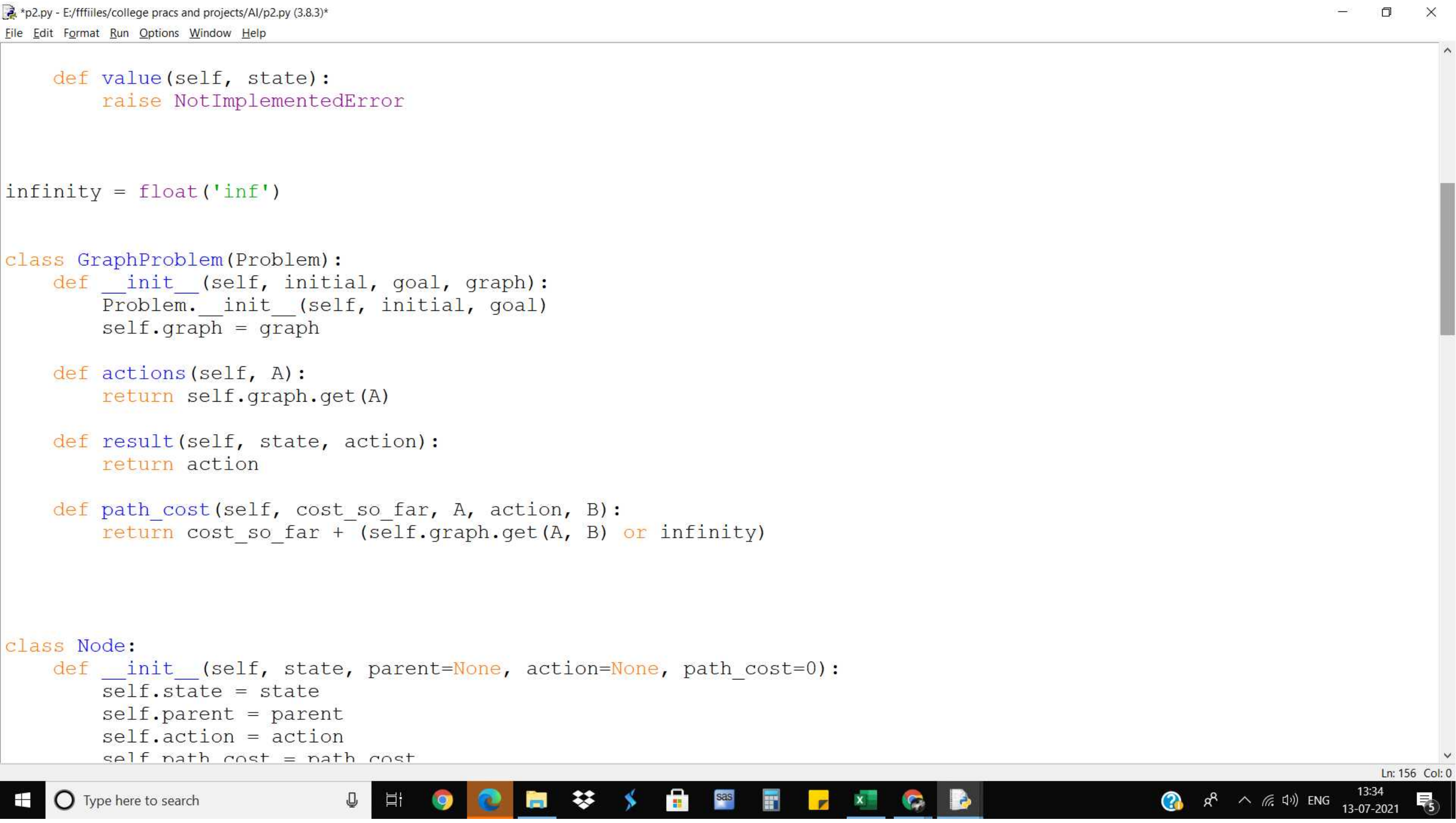
3) IDFS flowchart

IDDFS(tree):

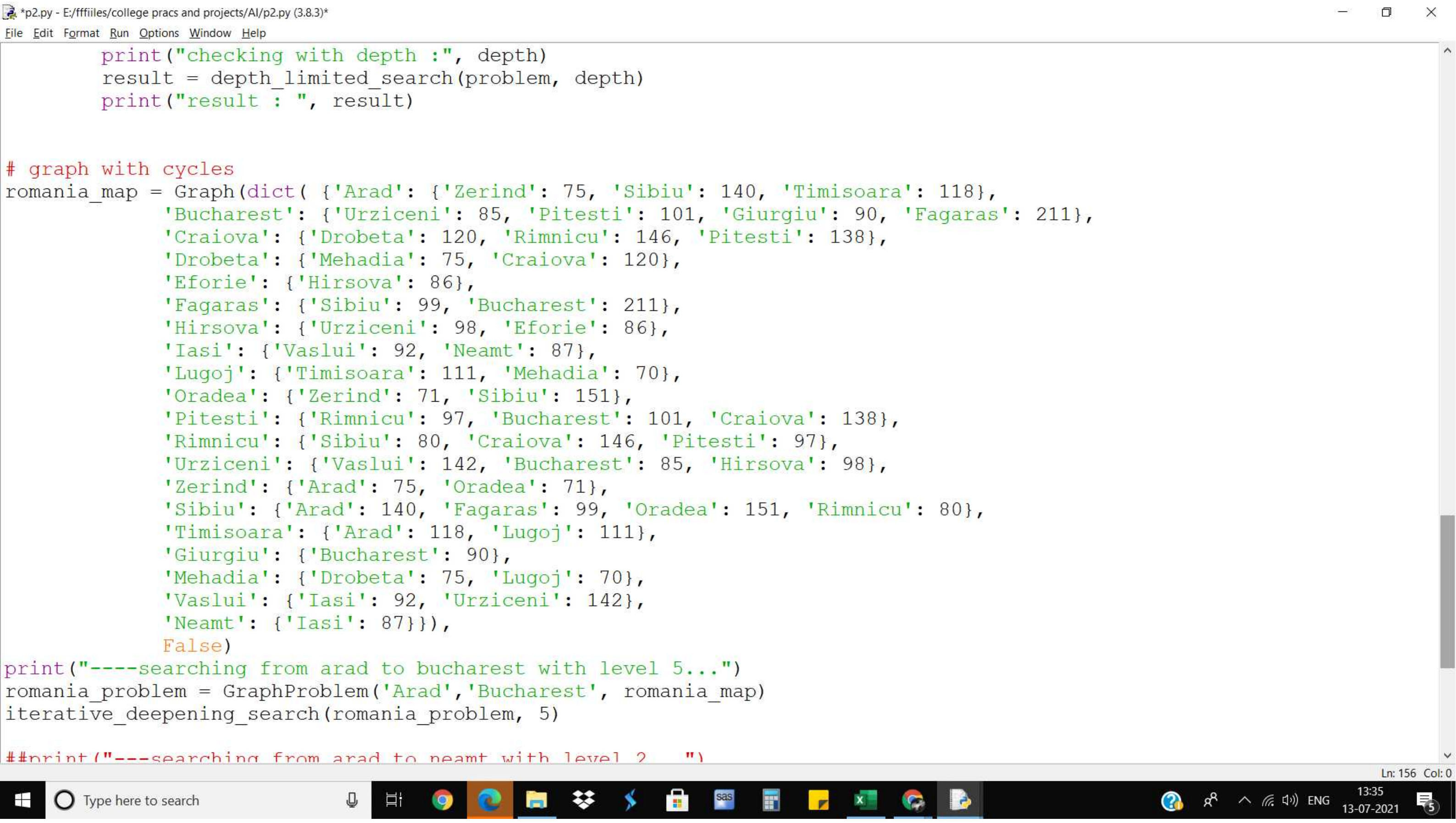




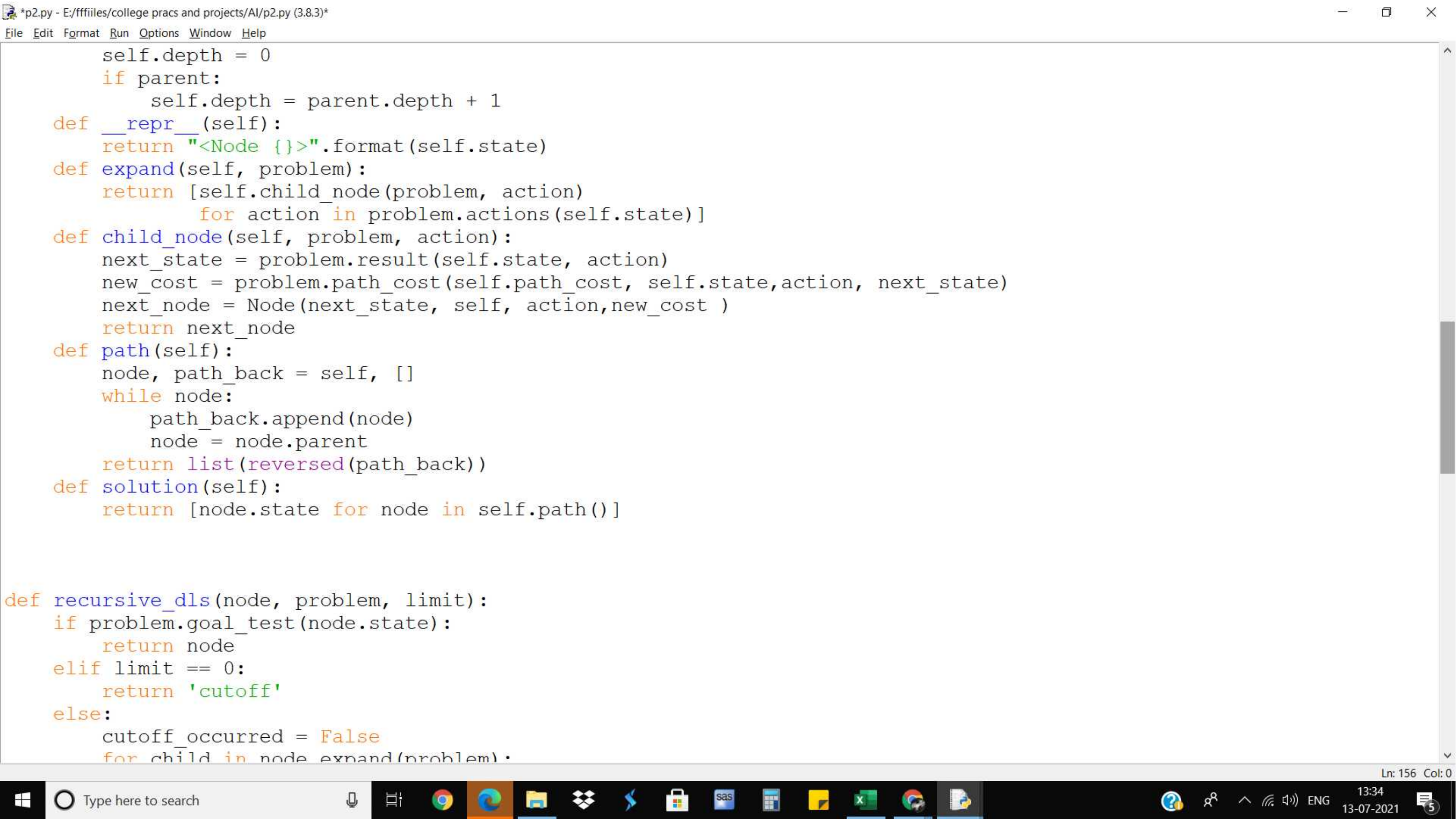
```
class Graph:
    def __init__(self, graph_dict=None, directed=True):
        self.graph_dict = graph_dict or {}
        self.directed = directed

    def get(self, a, b=None):
        links = self.graph_dict.setdefault(a, {})
        if b is None:
            return links
        else:
            return links.get(b)


class Problem(object):
    def __init__(self, initial, goal=None):
       self.initial = initial
       self.goal = goal

    def actions(self, state):
         raise NotImplementedError

    def result(self, state, action):
        raise NotImplementedError

    def goal_test(self, state):
            return state == self.goal

    def path_cost(self, c, state1, action, state2):
        return c + 1

    def value(self, state):
        raise NotImplementedError
```


p2.py - E:/fffiiles/college pracs and projects/AI/p2.py (3.8.3)
File Edit Format Run Options Window Help
Ln: 156 Col: 0
Type here to search
ENG
13:34
13-07-2021

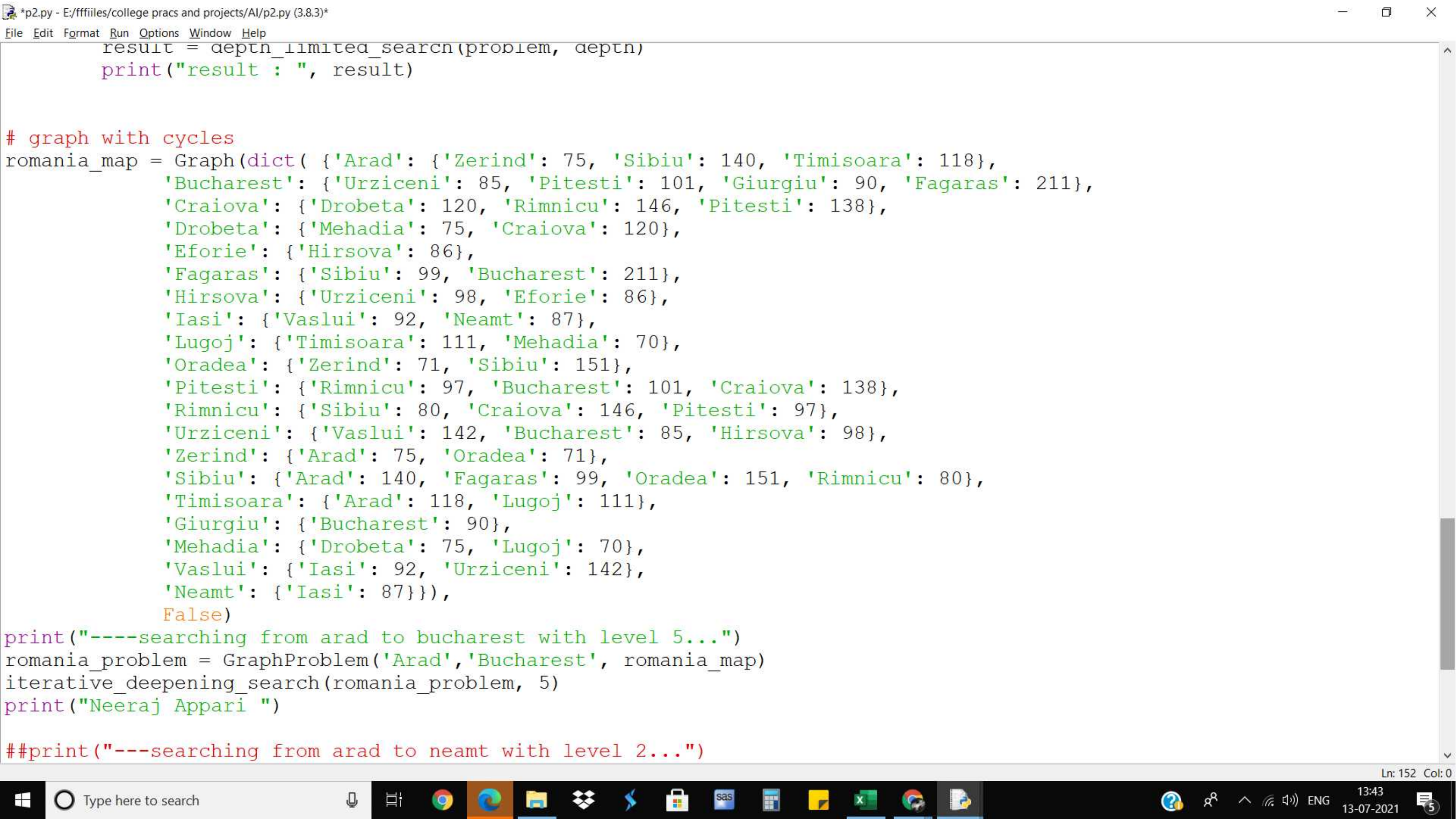
```
    def value(self, state):
        raise NotImplementedError



infinity = float('inf')


class GraphProblem(Problem):
    def __init__(self, initial, goal, graph):
        Problem.__init__(self, initial, goal)
        self.graph = graph

    def actions(self, A):
        return self.graph.get(A)

    def result(self, state, action):
        return action

    def path_cost(self, cost_so_far, A, action, B):
        return cost_so_far + (self.graph.get(A, B) or infinity)




class Node:
    def __init__(self, state, parent=None, action=None, path_cost=0):
        self.state = state
        self.parent = parent
        self.action = action
        self.path_cost = path_cost
```


p2.py - E:/fffiiles/college pracs and projects/AI/p2.py (3.8.3)
File Edit Format Run Options Window Help

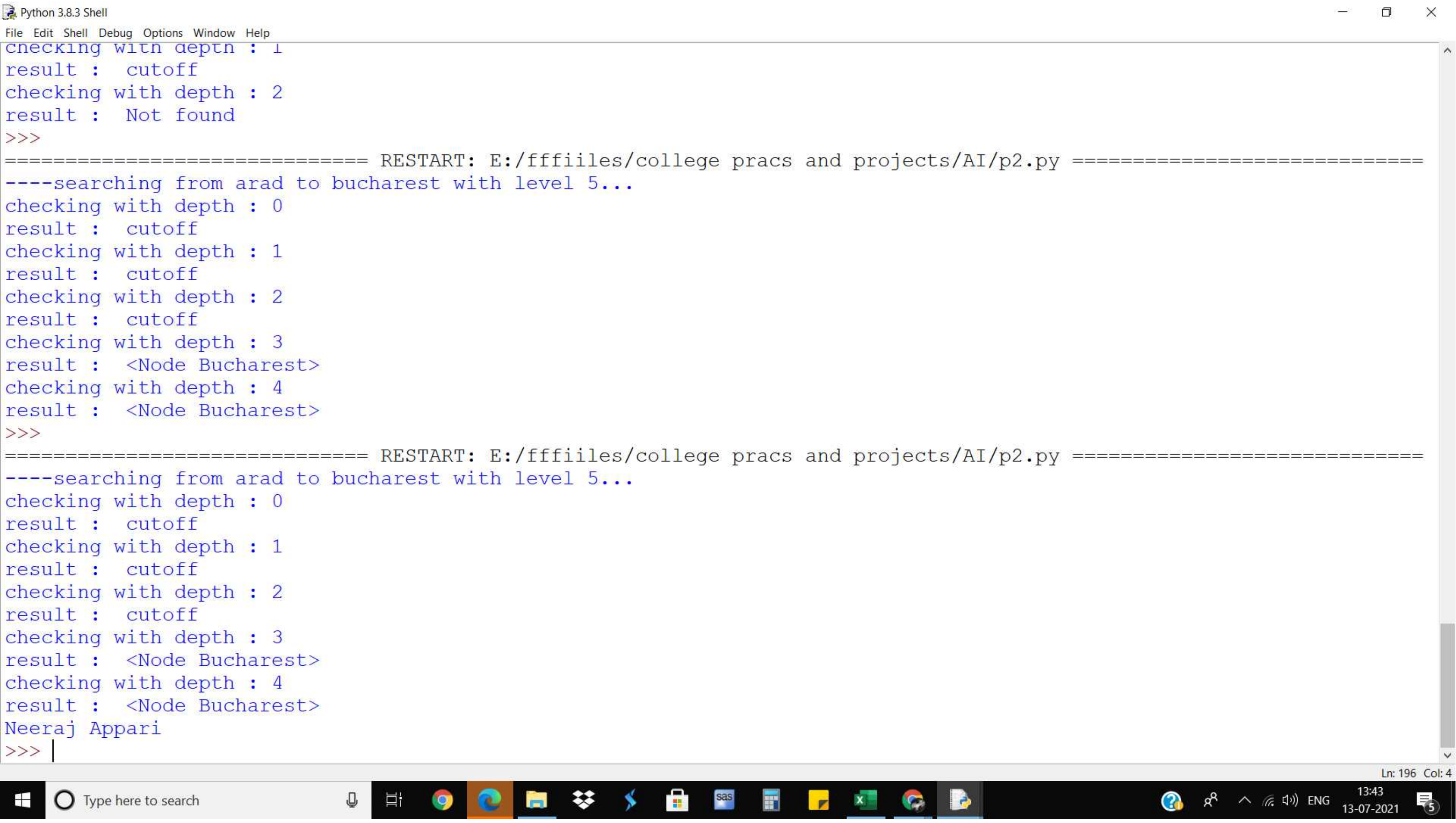
```
        print("checking with depth :", depth)
        result = depth_limited_search(problem, depth)
        print("result : ", result)


# graph with cycles
romania_map = Graph(dict( {'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
             'Bucharest': {'Urziceni': 85, 'Pitesti': 101, 'Giurgiu': 90, 'Fagaras': 211},
             'Craiova': {'Drobeta': 120, 'Rimnicu': 146, 'Pitesti': 138},
             'Drobeta': {'Mehadia': 75, 'Craiova': 120},
             'Eforie': {'Hirsova': 86},
             'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
             'Hirsova': {'Urziceni': 98, 'Eforie': 86},
             'Iasi': {'Vaslui': 92, 'Neamt': 87},
             'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
             'Oradea': {'Zerind': 71, 'Sibiu': 151},
             'Pitesti': {'Rimnicu': 97, 'Bucharest': 101, 'Craiova': 138},
             'Rimnicu': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
             'Urziceni': {'Vaslui': 142, 'Bucharest': 85, 'Hirsova': 98},
             'Zerind': {'Arad': 75, 'Oradea': 71},
             'Sibiu': {'Arad': 140, 'Fagaras': 99, 'Oradea': 151, 'Rimnicu': 80},
             'Timisoara': {'Arad': 118, 'Lugoj': 111},
             'Giurgiu': {'Bucharest': 90},
             'Mehadia': {'Drobeta': 75, 'Lugoj': 70},
             'Vaslui': {'Iasi': 92, 'Urziceni': 142},
             'Neamt': {'Iasi': 87}}),
             False)
print("----searching from arad to bucharest with level 5...")
romania_problem = GraphProblem('Arad','Bucharest', romania_map)
iterative_deepening_search(romania_problem, 5)

##print("---searching from arad to neamt with level 2...")
```



```
        self.depth = 0
        if parent:
            self.depth = parent.depth + 1
    def __repr__(self):
        return "<Node {}>".format(self.state)
    def expand(self, problem):
        return [self.child_node(problem, action)
                for action in problem.actions(self.state)]
    def child_node(self, problem, action):
        next_state = problem.result(self.state, action)
        new_cost = problem.path_cost(self.path_cost, self.state,action, next_state)
        next_node = Node(next_state, self, action,new_cost )
        return next_node
    def path(self):
        node, path_back = self, []
        while node:
            path_back.append(node)
            node = node.parent
        return list(reversed(path_back))
    def solution(self):
        return [node.state for node in self.path()]



def recursive_dls(node, problem, limit):
    if problem.goal_test(node.state):
        return node
    elif limit == 0:
        return 'cutoff'
    else:
        cutoff_occurred = False
        for child in node.expand(problem):
```


p2.py - E:/fffiles/college pracs and projects/AI/p2.py (3.8.3)
File Edit Format Run Options Window Help
Ln: 152 Col: 0
Type here to search
ENG
13:43
13-07-2021

```
        result = depth_limited_search(problem, depth)
        print("result : ", result)


# graph with cycles
romania_map = Graph(dict( {'Arad': {'Zerind': 75, 'Sibiu': 140, 'Timisoara': 118},
             'Bucharest': {'Urziceni': 85, 'Pitesti': 101, 'Giurgiu': 90, 'Fagaras': 211},
             'Craiova': {'Drobeta': 120, 'Rimnicu': 146, 'Pitesti': 138},
             'Drobeta': {'Mehadia': 75, 'Craiova': 120},
             'Eforie': {'Hirsova': 86},
             'Fagaras': {'Sibiu': 99, 'Bucharest': 211},
             'Hirsova': {'Urziceni': 98, 'Eforie': 86},
             'Iasi': {'Vaslui': 92, 'Neamt': 87},
             'Lugoj': {'Timisoara': 111, 'Mehadia': 70},
             'Oradea': {'Zerind': 71, 'Sibiu': 151},
             'Pitesti': {'Rimnicu': 97, 'Bucharest': 101, 'Craiova': 138},
             'Rimnicu': {'Sibiu': 80, 'Craiova': 146, 'Pitesti': 97},
             'Urziceni': {'Vaslui': 142, 'Bucharest': 85, 'Hirsova': 98},
             'Zerind': {'Arad': 75, 'Oradea': 71},
             'Sibiu': {'Arad': 140, 'Fagaras': 99, 'Oradea': 151, 'Rimnicu': 80},
             'Timisoara': {'Arad': 118, 'Lugoj': 111},
             'Giurgiu': {'Bucharest': 90},
             'Mehadia': {'Drobeta': 75, 'Lugoj': 70},
             'Vaslui': {'Iasi': 92, 'Urziceni': 142},
             'Neamt': {'Iasi': 87}}),
             False)
print("----searching from arad to bucharest with level 5...")
romania_problem = GraphProblem('Arad','Bucharest', romania_map)
iterative_deepening_search(romania_problem, 5)
print("Neeraj Appari ")

##print("---searching from arad to neamt with level 2...")
```



```
checking with depth : 1
result :  cutoff
checking with depth : 2
result :  Not found
>>>
============================== RESTART: E:/fffiiles/college pracs and projects/AI/p2.py ==============================
----searching from arad to bucharest with level 5...
checking with depth : 0
result :  cutoff
checking with depth : 1
result :  cutoff
checking with depth : 2
result :  cutoff
checking with depth : 3
result :  <Node Bucharest>
checking with depth : 4
result :  <Node Bucharest>
>>>
============================== RESTART: E:/fffiiles/college pracs and projects/AI/p2.py ==============================
----searching from arad to bucharest with level 5...
checking with depth : 0
result :  cutoff
checking with depth : 1
result :  cutoff
checking with depth : 2
result :  cutoff
checking with depth : 3
result :  <Node Bucharest>
checking with depth : 4
result :  <Node Bucharest>
Neeraj Appari
>>>
```

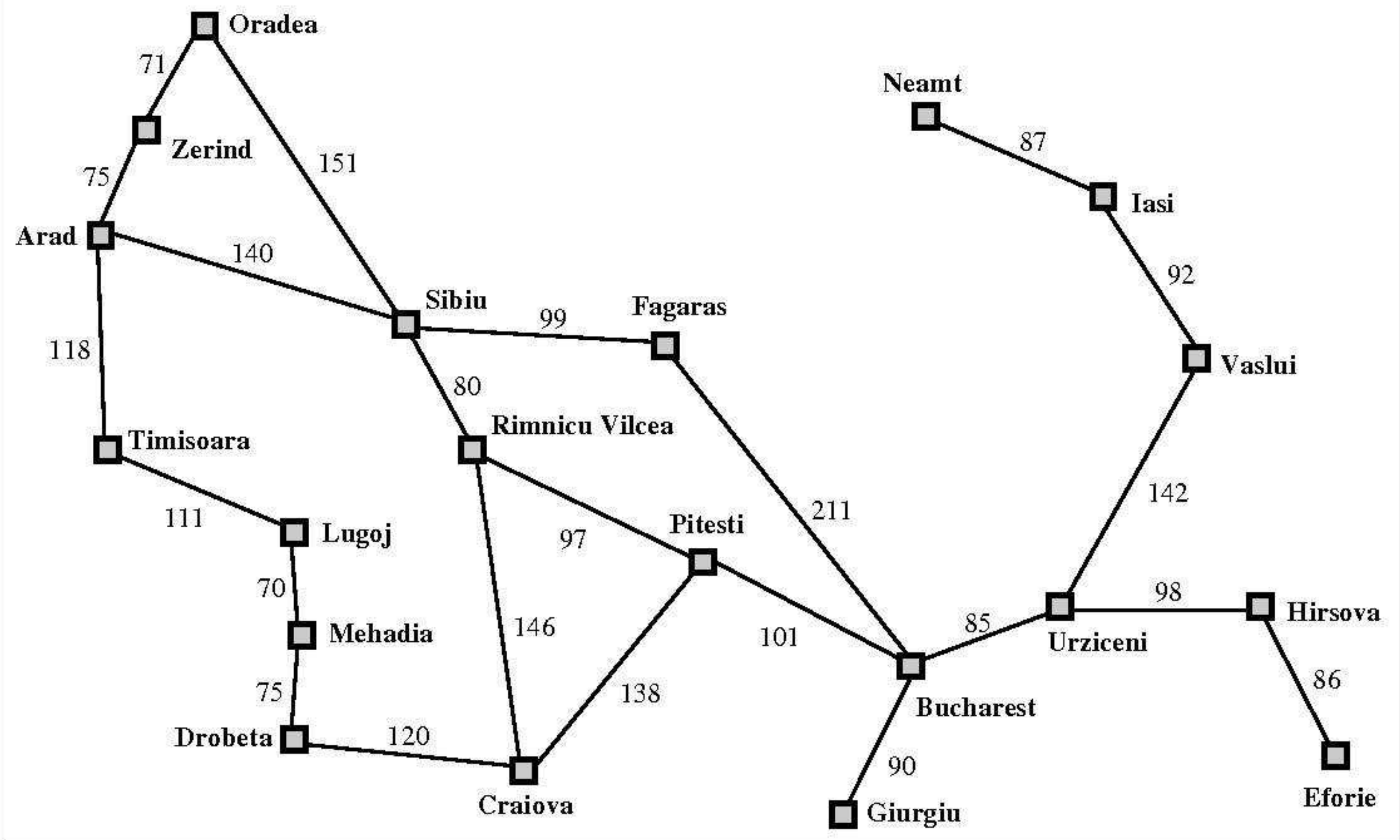


Figure 3.2 A simplified road map of part of Romania.